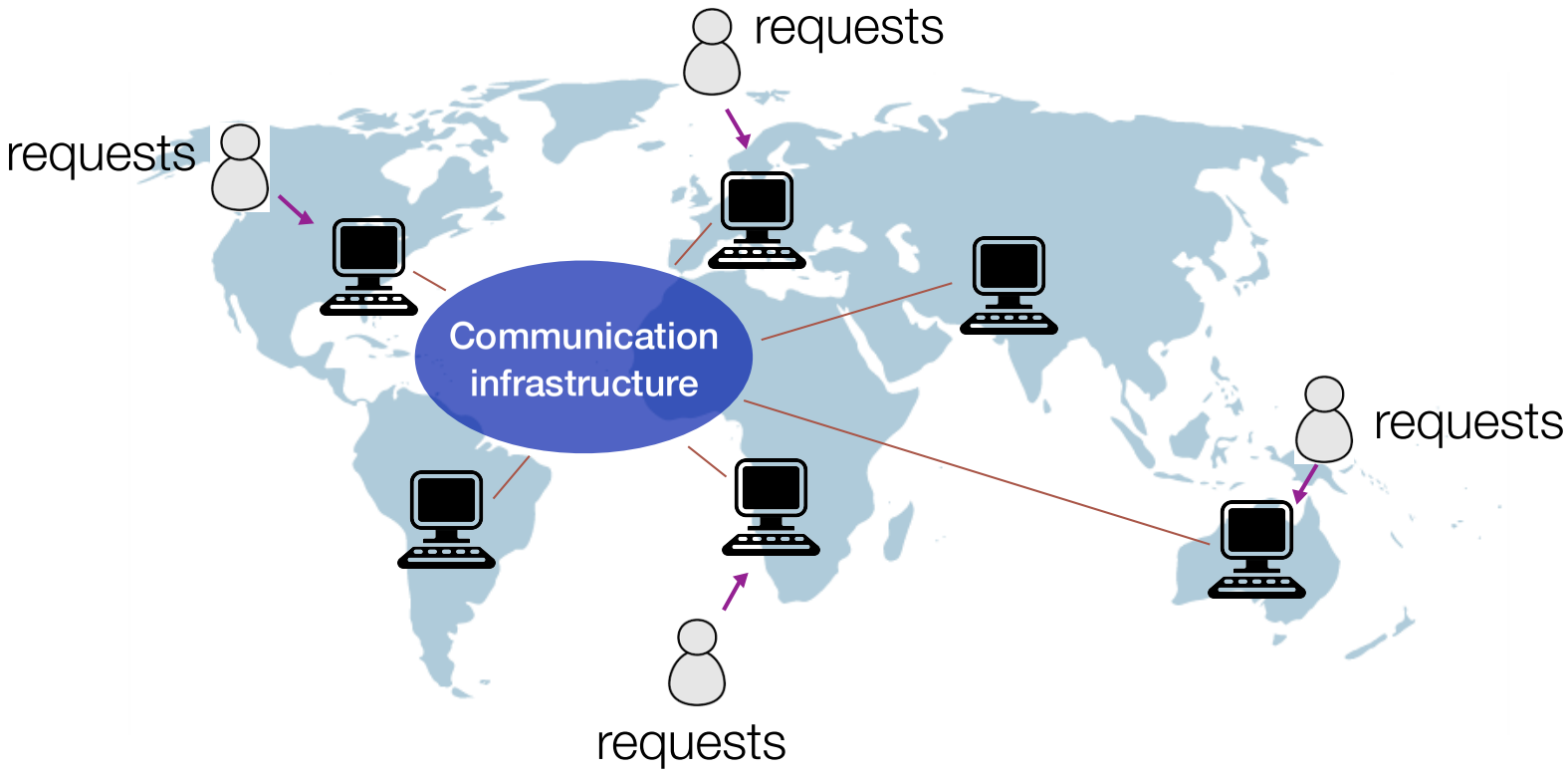
Automated Testing of Distributed Systems Against Functional Specifications

Constantin Enea

Ecole Polytechnique, LIX

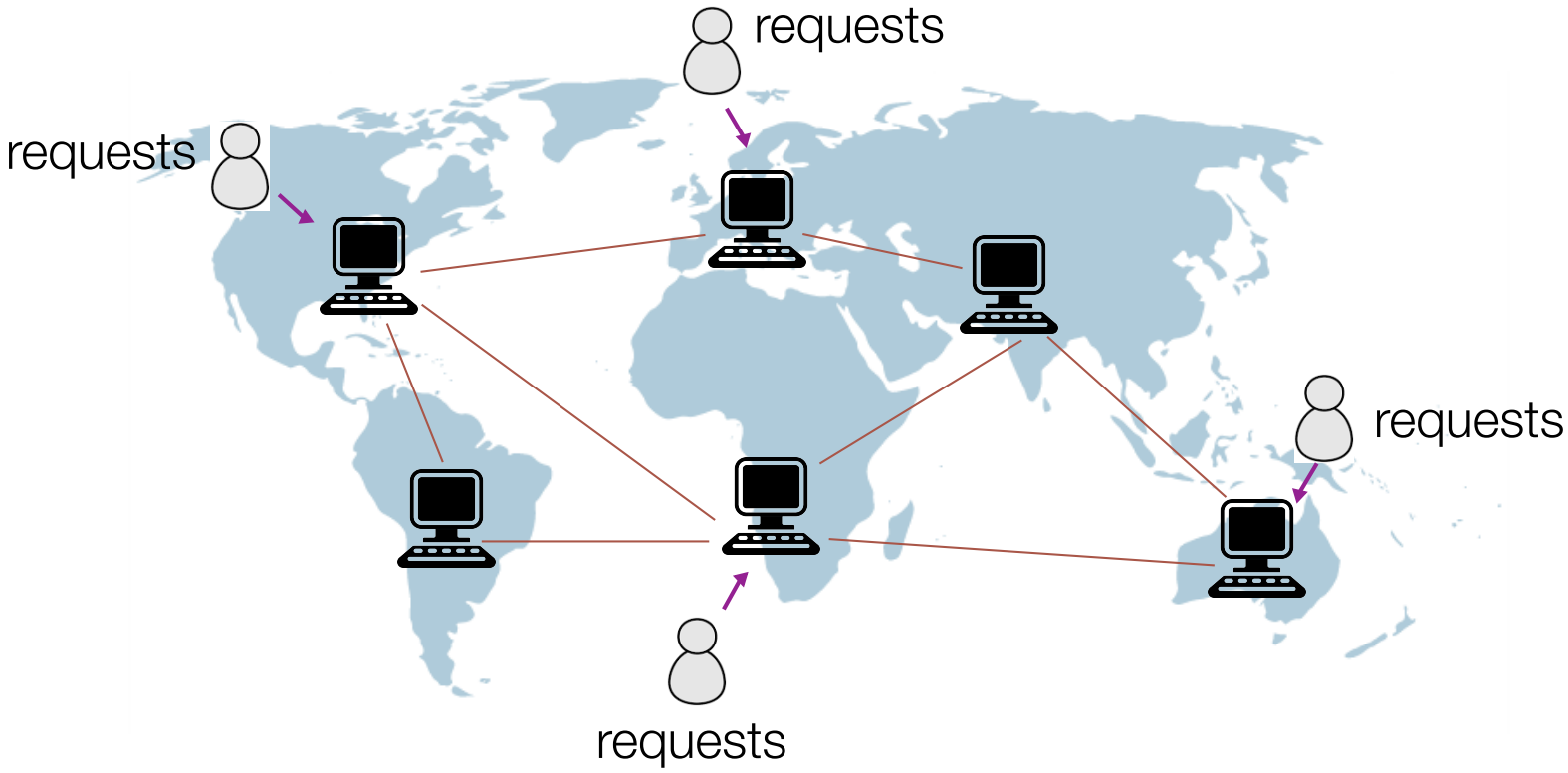
Distributed Systems



Nodes that collaborate to ensure a service to a large number of widely spread users

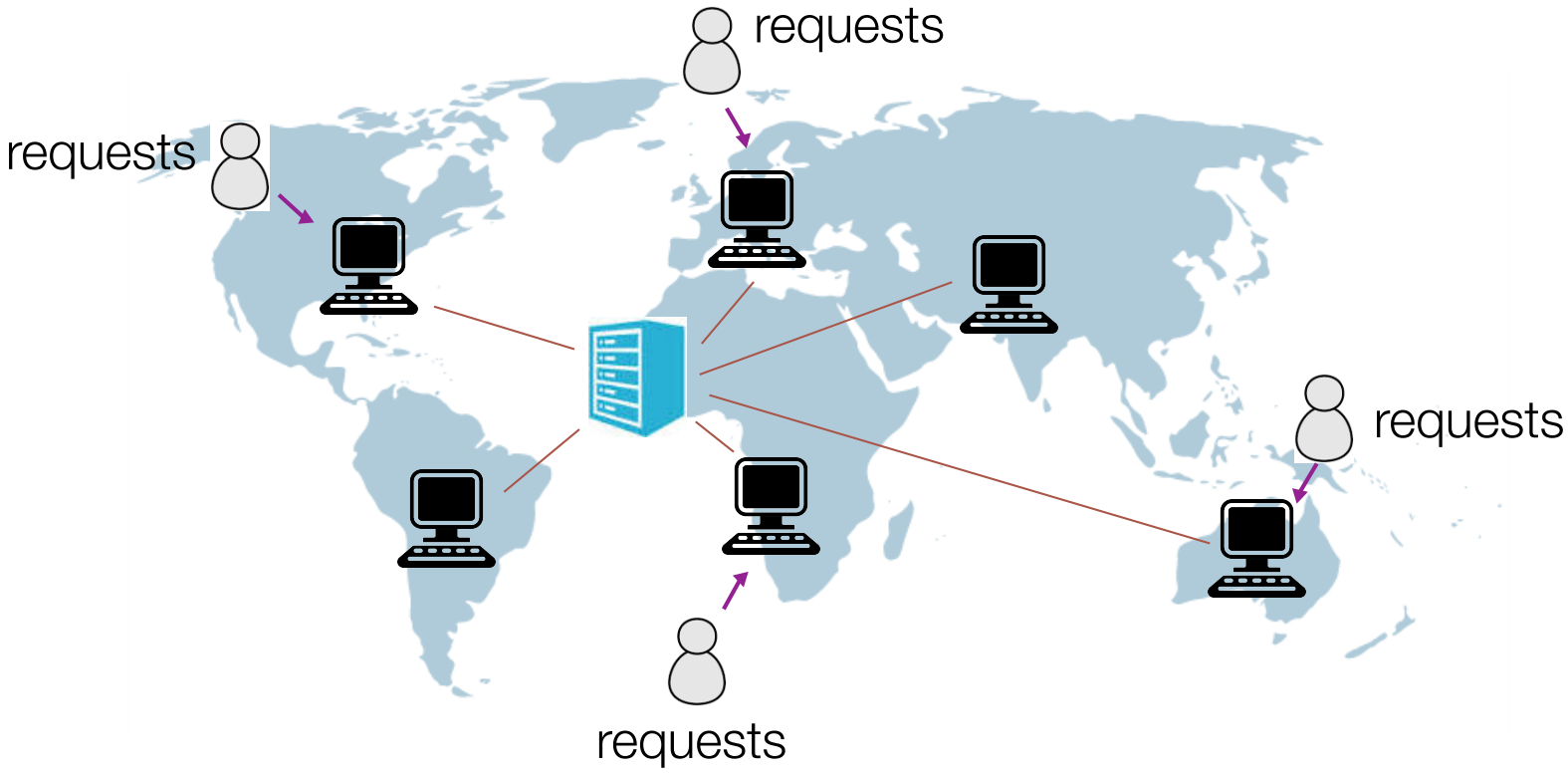
(tolerates faults and asynchrony)

Distributed Systems



Message Passing Communication

Distributed Systems



Communicating using a (distributed) shared state/memory

database

(Unit) Testing Distributed Systems

Effectiveness: high probability of exposing bugs

Interpretability: ability to find the root-cause of a bug in an execution

(Unit) Testing Distributed Systems

Effectiveness: high probability of exposing bugs

Interpretability: ability to find the root-cause of a bug in an execution

Seemingly opposing requirements:

- effectiveness needs **many** faults, **a lot** of asynchrony, **big** workloads if we are using the runtime
- interpretability needs “simple” executions, **small** workloads, **less** faults and asynchrony

(Unit) Testing Distributed Systems

Effectiveness: high probability of exposing bugs

Interpretability: ability to find the root cause of a bug in an execution

Seem

- effe
  wo
- interpretability needs “simple” executions, **small** workloads, **less** faults and asynchrony

Ensuring effectiveness **and** interpretability**:** introducing faults and asynchrony in a **systematic manner**

Plan

1. **Testing consensus** protocol implementations

   [Drăgoi, E, Ozkan, Majumdar, Niksic, OOPSLA'20]

2. **Testing database-backed applications**

   [Biswas, Kakwani, Vedurada, E, Lal, OOPSLA'21]

Plan

1. **Testing consensus** protocol implementations

   [Drăgoi, E, Ozkan, Majumdar, Niksic, OOPSLA'20]

2. **Testing database-backed applications**

   [Biswas, Kakwani, Vedurada, E, Lal, OOPSLA'21]

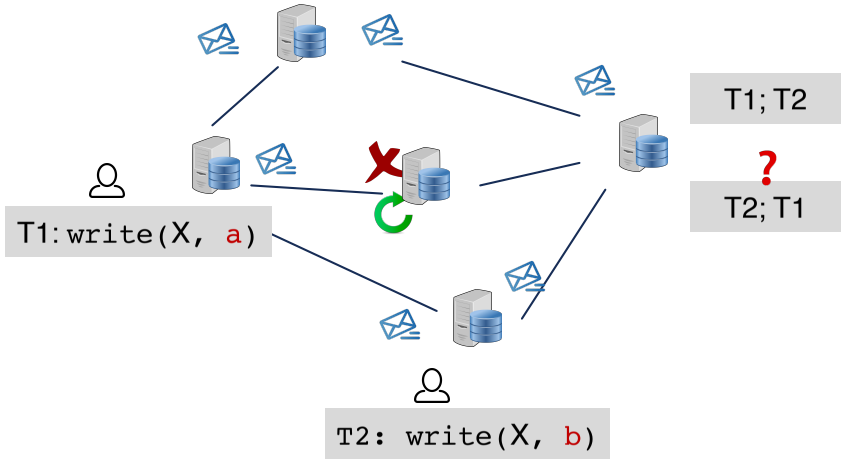
Consensus Protocols

At the heart of many distributed systems

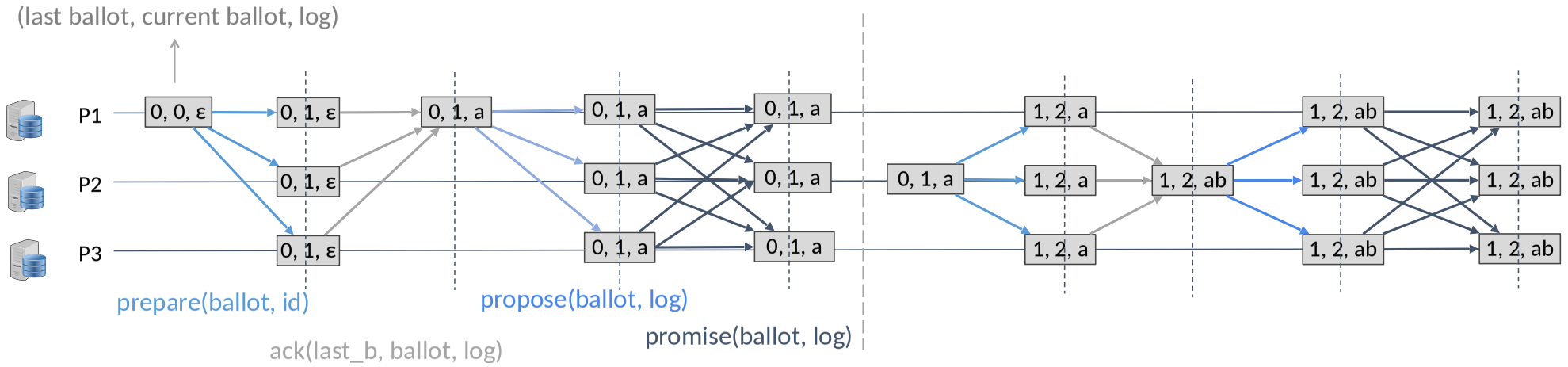
Provides agreement among of set of nodes

- message-passing communication
- network/node faults

Examples: Paxos, ViewStamped, Raft, etc.



An Example of Consensus Algorithm



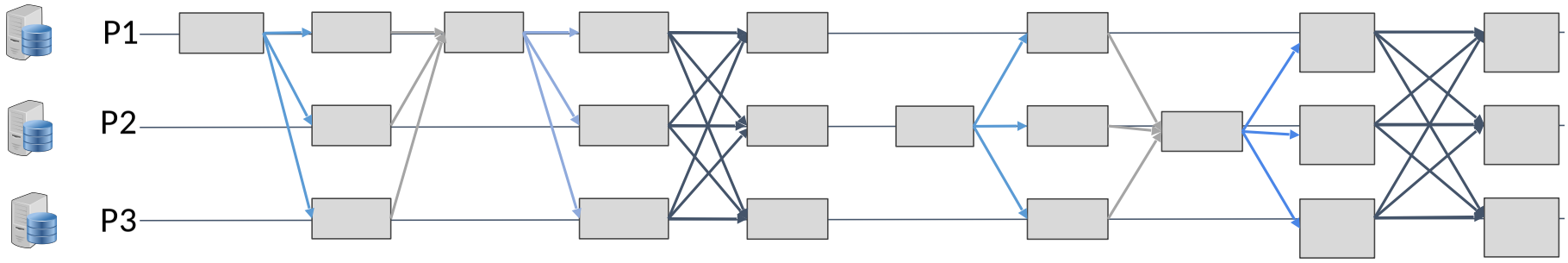
A sequence of **rounds**

- in a round: send messages + receive messages and update state

Should behave correctly in the presence of asynchrony, network link failures, node failures

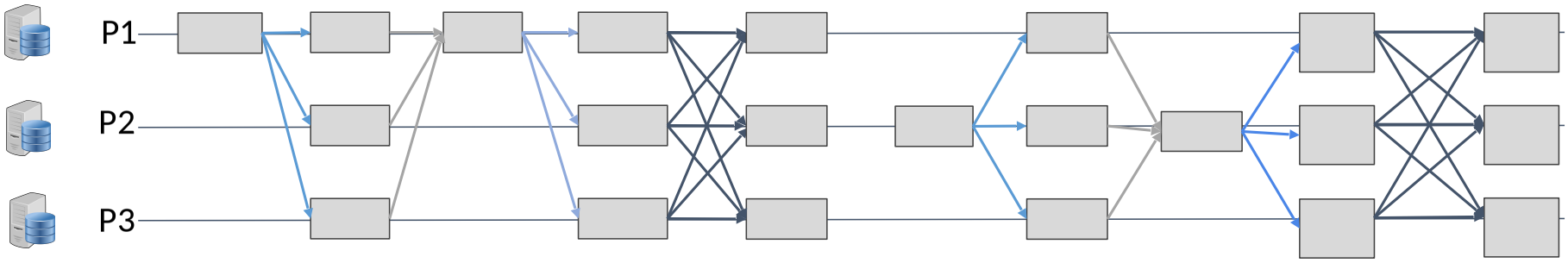
Many possible executions

An execution with no message delays, drops, network partitions, etc.

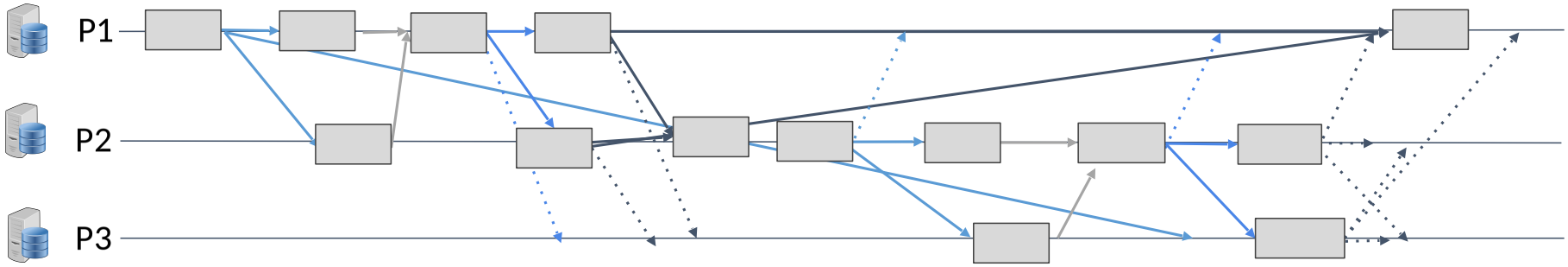


Many possible executions

An execution with no message delays, drops, network partitions, etc.

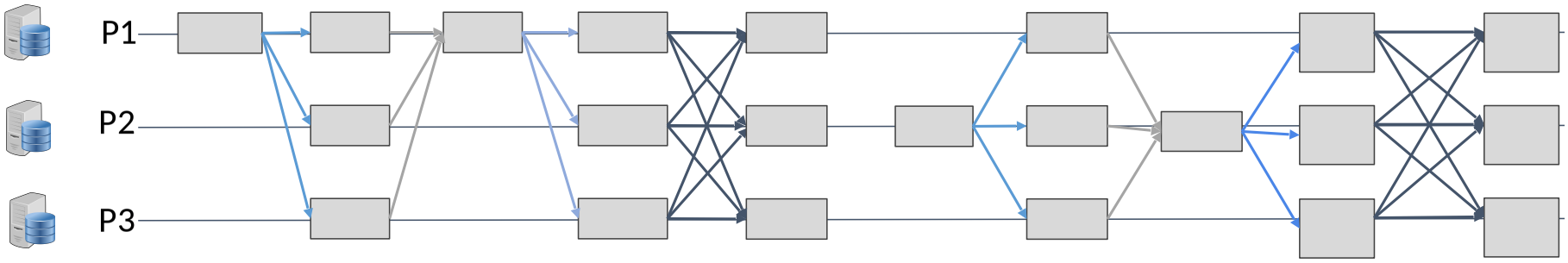


An execution with message delays, drops, network partitions, etc.

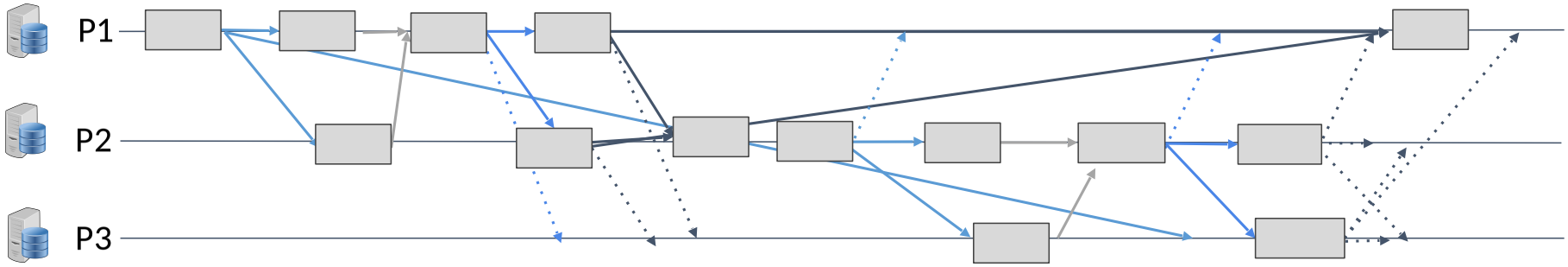


Many possible executions

An execution with no message delays, drops, network partitions, etc.



An execution with message delays, drops, network partitions, etc.



Incorrect implementations may cause bugs in subtle executions

Contribution

Randomized testing algorithm that exploits semantic properties of consensus protocols to reduce the space of executions it enumerates

Contribution

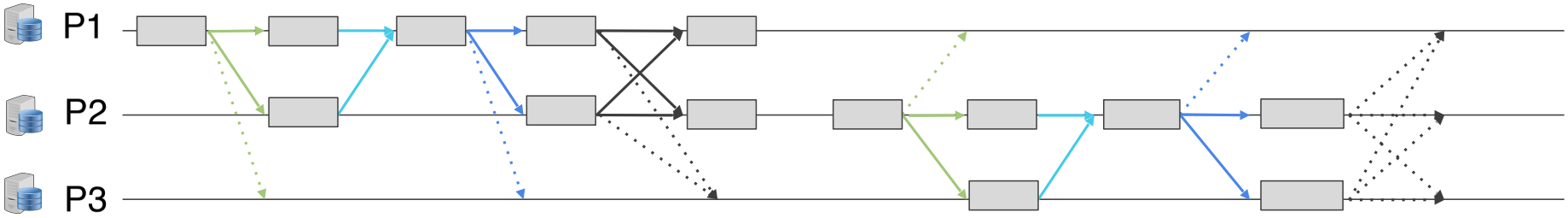
Randomized testing algorithm that exploits semantic properties of consensus protocols to reduce the space of executions it enumerates

Exploits communication closure of consensus protocols

Samples from synchronous executions

- semantic reduction of the execution space (effectiveness)
- provides executions that are easier to debug (interpretability)

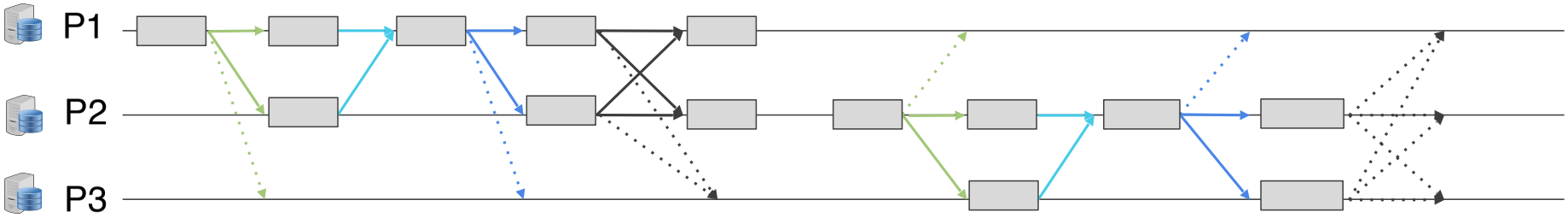
Communication Closure



Lossy synchronous executions: a number of communication-closed rounds

- in a round: send messages + receive messages and update the state
- rounds are executed in a lockstep manner
- messages are delivered in the round they are sent or otherwise, discarded

Communication Closure



Lossy synchronous executions: a number of communication-closed rounds

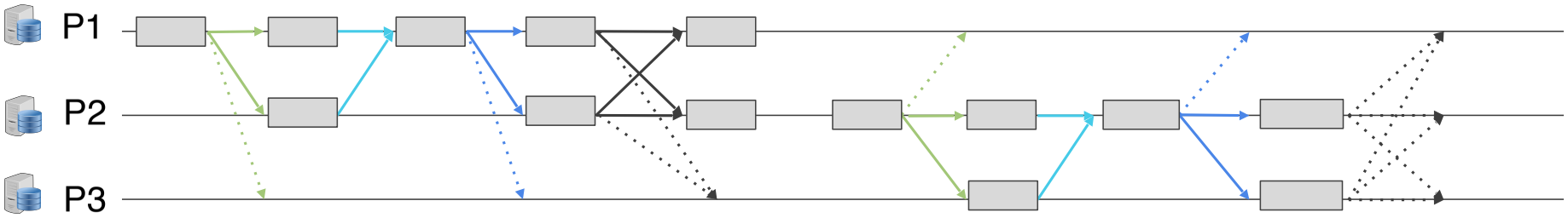
- in a round: send messages + receive messages and update the state
- rounds are executed in a lockstep manner
- messages are delivered in the round they are sent or otherwise, discarded

Standard consensus protocols are **communication-closed**: every execution is equivalent to a lossy synchronous one

Communication Closure Hypothesis for Testing

Bugs already manifest in uniform lossy synchronous executions

isolating a set of processes (instead of dropping individual messages)

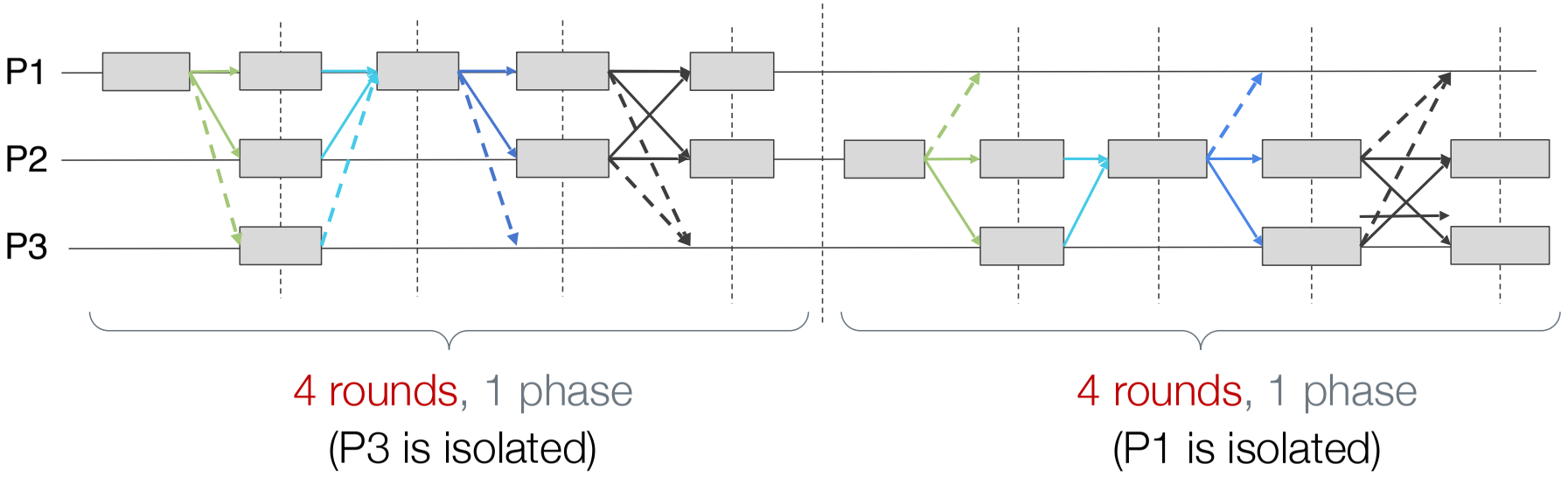


Randomized Testing

Samples from uniform lossy synchronous executions

Prioritizes the search space of executions based on:

- The number of process isolations: d
- The rate at which the failures are recovered: k



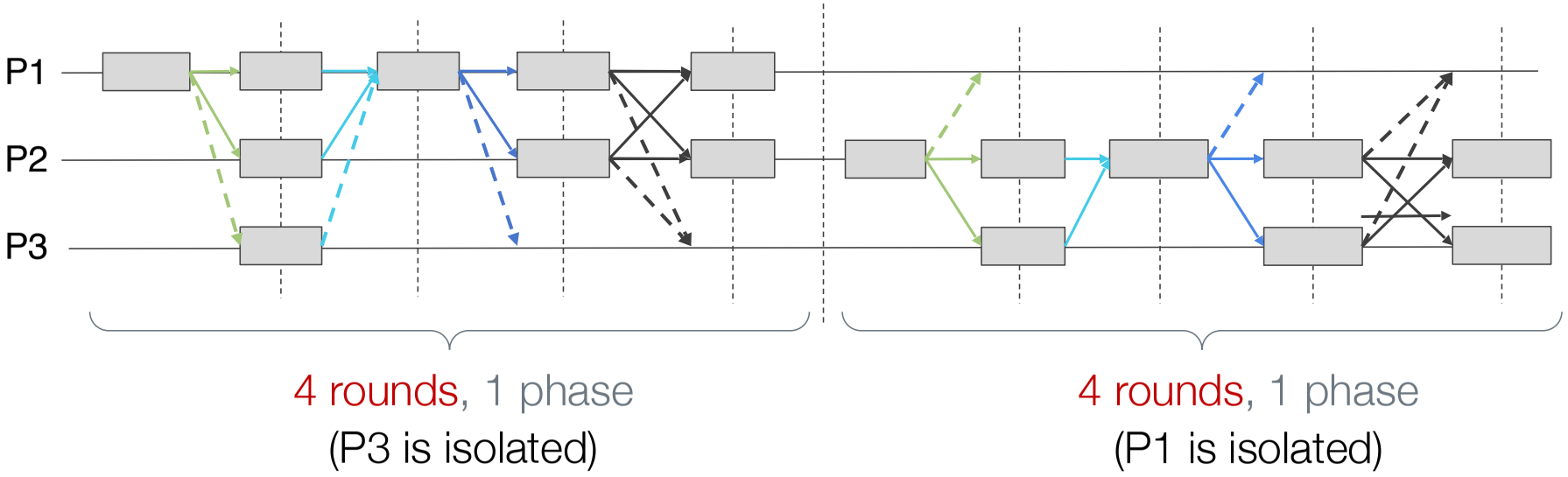
An example 2-bounded 4-periodic execution

Randomized Testing

Samples from d-bounded k-periodic uniform lossy synchronous executions

Prioritizes the search space of executions based on:

- The number of process isolations: d
- The rate at which the failures are recovered: k



An example 2-bounded 4-periodic execution

Experiments on Large-Scale Systems

Cassandra v2.0.0 – heavy instrumentation to enforce synchronized rounds

- Reproduced a known difficult bug: violation to serializability
- n=3 processes, p=4 phases (r=24 rounds, period k=6), #faults d in [5, 10]

Ratis v0.0.6 – lightweight instrumentation to identify rounds of messages

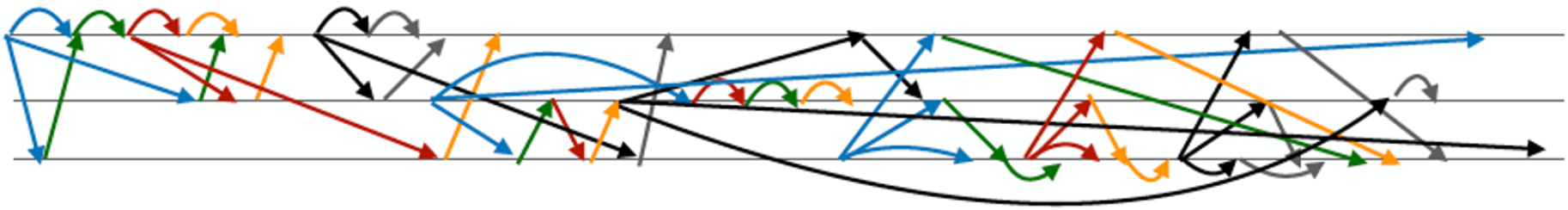
- Detected three new bugs: failure to respond to client, failure to elect a leader, failure to synchronize replicas
- n=3 processes, p=4 phases (r=8 rounds, period k=2), #faults d in [1, 7]

Zookeeper v3.5.8 – no instrumentation: abstract phases and rounds

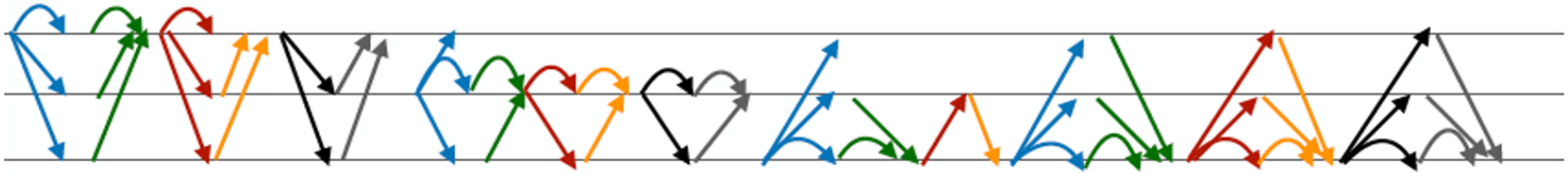
- Detected two new bugs along with a known bug: violation to sequential consistency, dropped client, and divergence
- n=3 processes, p=3 phases, #faults d in [3, 9]

Improving Interpretability

Trace sampled with “asynchronous” randomized sampling (for Cassandra)



Trace sampled with our algorithm



Plan

1. **Testing consensus** protocol implementations

   [Drăgoi, E, Ozkan, Majumdar, Niksic, OOPSLA'20]

2. **Testing database-backed applications**

   [Biswas, Kakwani, Vedurada, E, Lal, OOPSLA'21]

Bank Payment App

```
1 fn pay_for(acc_id, amount):
2   balance = read_account(acc_id);
3   if amount <= balance {
4     balance -= amount;
5     update_account(acc_id, balance);
6   }
```

```
1 fn pay_for(acc_id, amount):
2   balance = read_account(acc_id);
3   if amount <= balance {
4     balance -= amount;
5     update_account(acc_id, balance);
6   }
```

Possible double spending

Avoid interference ⇒ Transaction Isolation

Serializability

```
1 fn pay_for(acc_id, amount):
2   balance = read_account(acc_id);
3   if amount <= balance {
4     balance -= amount;
5     update_account(acc_id, balance);
6   }
```

```
1 fn pay_for(acc_id, amount):
2   balance = read_account(acc_id);
3   if amount <= balance {
4     balance -= amount;
5     update_account(acc_id, balance);
6   }
```

Weakening Serializability

```
1 fn pay_for(acc_id, amount):
2   balance = read_account(acc_id);
3   if amount <= balance {
4     balance -= amount;
5     update_account(acc_id, balance);
6   }
```

```
1 fn pay_for(acc_id, amount):
2   balance = read_account(acc_id);
3   if amount <= balance {
4     balance -= amount;
5     update_account(acc_id, balance);
6   }
```

Serializability vs Snapshot Isolation

Isolation Levels

Performance vs Guarantees ⇒ multiple isolation levels

Checking correctness under a certain isolation level

- Bank Payment is correct under Serializability, Snapshot Isolation, but fails under Read Committed

Testing Coverage: Production Databases

Forcing “weak” behaviors (non serializable) requires big workloads and ad-hoc manipulation of the setup (inject network faults)

Sensitive to a particular implementation of an isolation level

Challenge

Ensuring coverage with small workloads

Being agnostic to different setups and implementations of same isolation levels

Contribution

MonkeyDB

A mock database, reference implementation of isolation levels

- Effective testing with small workloads
- Key-Value and SQL interface (SQL compiler to Key-Value)
- In memory database, no network manipulation

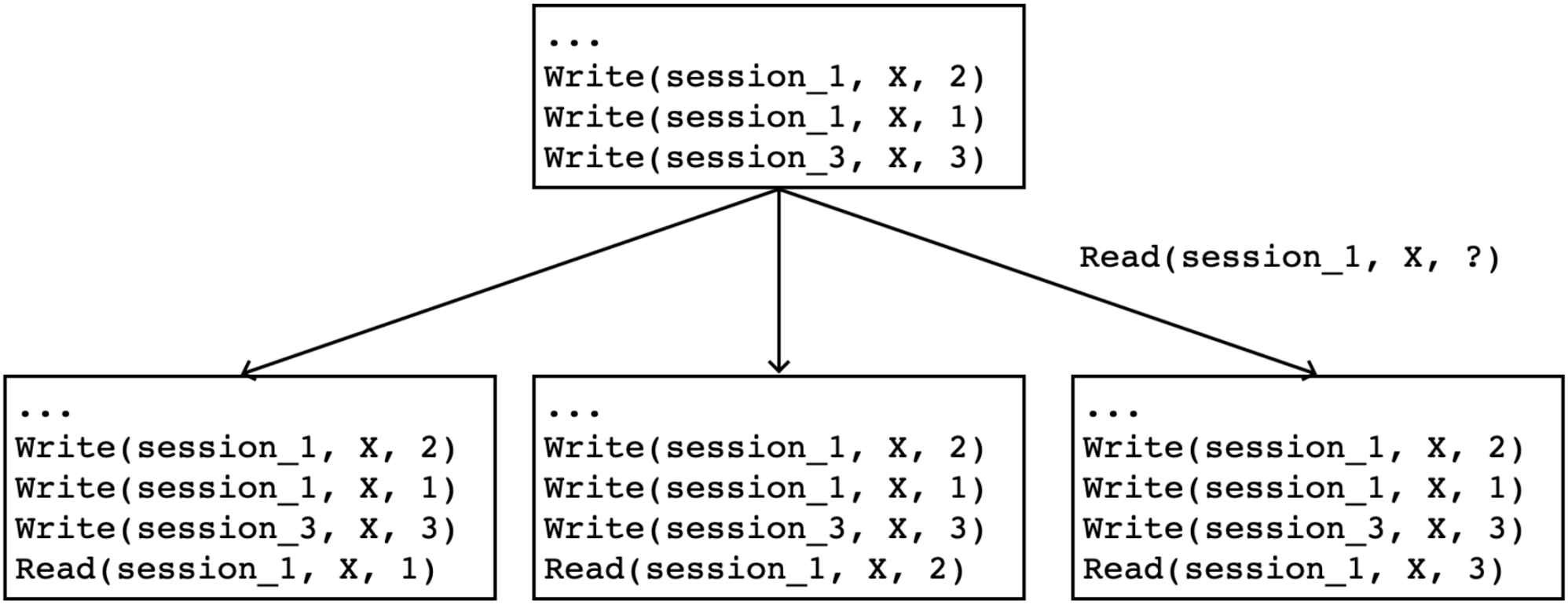
Implementation

- A log of reads and writes as storage
- Reads can return “old” values
- Logs are checked to satisfy the considered isolation level, using a formal axiomatic semantics [Biswas, E, OOPSLA’19]

Implementation: Writes

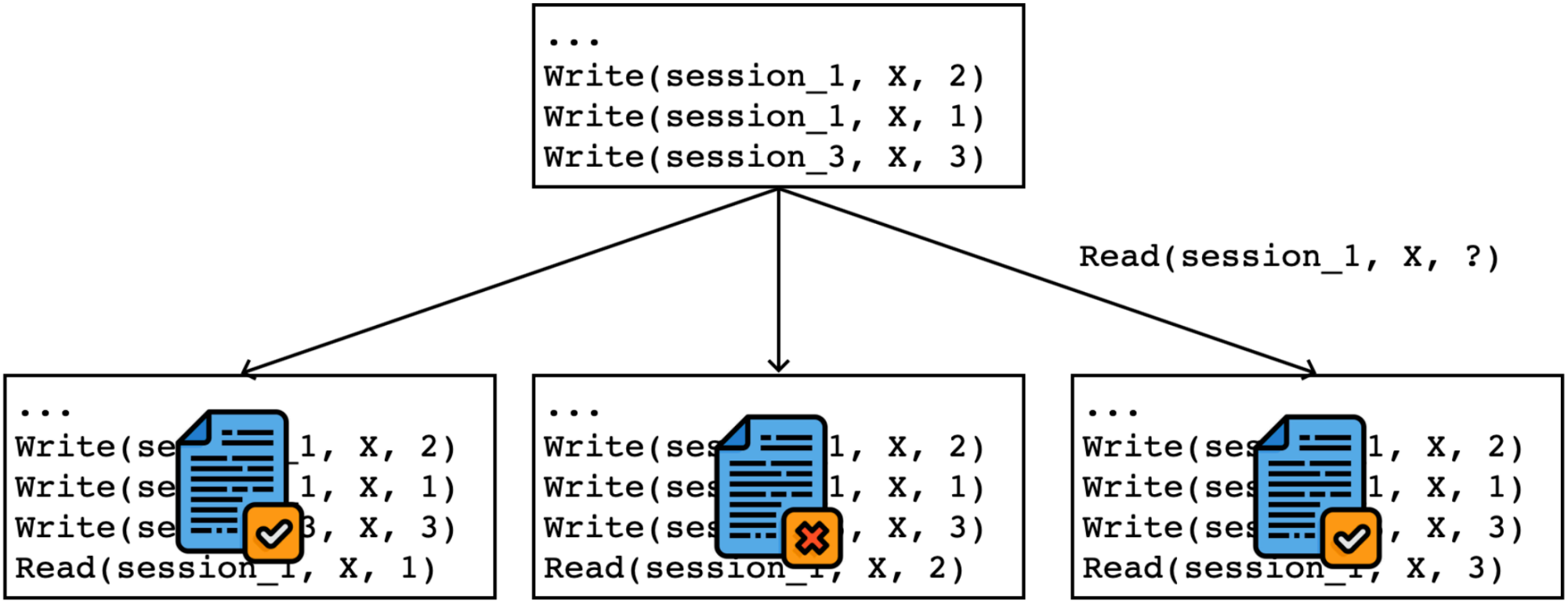
Writes are simply appended to the log

Implementation: Reads



Compute possible logs for a read

Implementation: Reads



Compute possible logs for a read

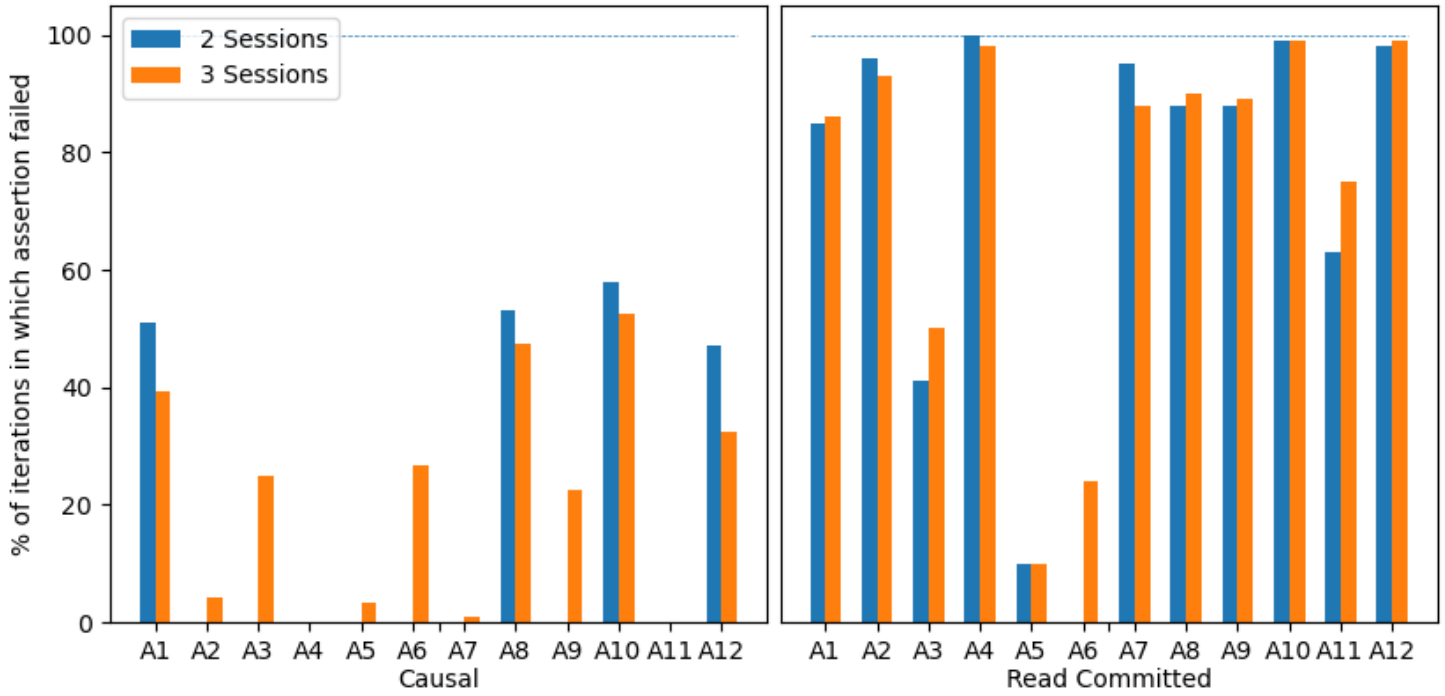
Check validity based on an axiomatic model

Filter out the valid ones and select one randomly

Experimental Evaluation

Benchmark: a subset of OLTPBench

TPC-C: testing for 12 invariants extracted from its specification (that hold under SER)



Effective in breaking assertions (% out of 100 iterations) - running with MySQL did **not** violate any assertion except A10 and A12 (even with 10 sessions).

Conclusions

Randomized testing techniques that are effective and simplify debugging

- message passing communication or storage-backed communication
- based on formal models of executions (semantics)
- systematizing fault introduction and asynchrony

Future work:

- domain specific languages to specify restrictions to subsets of executions
- reinforcement learning for exploring the execution space